

# Logical Control for Mobile Robots

Tarek M. Sobh, Mohamed Dekhil, Alyosha A. Efros and Raul Mihalí

Computer Science Department

University of Utah

Salt Lake City, Utah 84112, USA

## Abstract

In this work we present a distributed sensor-based control strategy for mobile robot navigation. We investigate a server-client model, where the clients are executing their tasks in parallel. The logical sensor approach is used as a hybrid framework to model and implement the sensory system for control of the mobile robot. The framework allows for a hierarchical data representation scheme, where sensory data and uncertainty is modeled and used at different levels, depending on the nature of the requested control command.

**Keywords:** Mobile Robots, Uncertainty Modeling, Distributed Control, Sensing.

## 1 Introduction

In any closed-loop control system, sensors are used to provide the feedback information that represents the current status of the system and the environmental uncertainties. The main component in such systems is the transformation of sensor outputs to the decision space, then

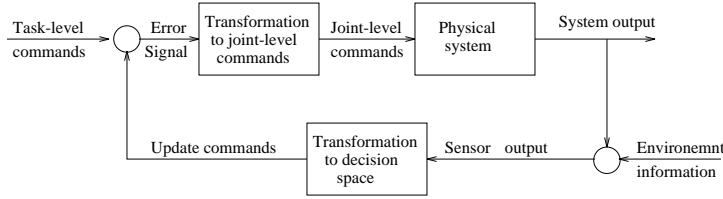


Figure 1: Closed loop control system.

the computation of the error signals and the joint-level commands (see Figure 1). For example, the sensor readings might be the current tool position, the error signal the difference between the desired and current position at this moment, and finally, the joint-level command will be the required actuator torque/force.

The sensors used in the control scheme shown in Figure 1 are considered to be passive elements that provide raw data to a central controller. The central controller computes the next command based on the required task and the sensor readings. The disadvantage of this scheme is that the central controller may become a bottleneck when the number of sensors increases which may lead to longer response time. By response time we mean the time between two consecutive commands. In some applications the required response time may vary according to the required task and the environment status. For example, in autonomous mobile robot with the task of reaching a destination position while avoiding unknown obstacles, the time to reach to the required position may not be important, however, the response time for avoiding obstacles is critical and requires fast response.

Fast response can be achieved by allowing sensors to send commands directly to the physical system when quick attention is required. This is analogous to human reactions to some events. In the normal cases, the sensory systems in humans (e.g., eye, ear, nerves, etc.) sends perceived

data to the brain (the central controller) which analyze this data and decides the next action to be taken based on the result of the analysis and the required task to be done. However, humans have a very fast contracting reaction when touching hot surfaces for example. In such cases, this reaction behavior is due to commands sent directly from the nerves at the skin spot where the touch occurred to the muscles, bypassing the brain. This particular type of feedback control and response needs to be encapsulated in sensing controllers.

In this work, several controllers (clients) are working in parallel, competing for the server. The server selects the command to be executed based on a dynamically configured priority scheme. Each of these clients has a certain task, and can use the sensor readings to achieve its goal. A special client with the task of avoiding obstacles is assigned the highest priority. The clients need to know the current state of the system and the command history to update their control strategy. Therefore, the server has to broadcast the selected command and the current state of the system. Commands with lower priorities will be discarded and their sender sensors notified. Tasks with the same priorities will be randomly sampled, unless a strictly ordered priority function is being considered.

Another aspect of this work is incorporating tolerance analysis and measures into the used sensory system. This provides quantitative measures for the accuracy of the location of measured points. It also serves as the basis for devising sensing strategies to enhance the measured data for localization and map construction.

The logical sensor approach, which we used to model the sensory system in our mobile robot, allows flexible and modular design of the controllers. It also provides several levels of data abstraction and tolerance analysis based on the sensor type and the required task. The

initial work on this project is described in [1]. This approach is used to build high-level requests which may be used by the application program. These requests include measuring data points within a specific tolerance or within a certain time limit.

A brief background and related work in sensor-based control and mobile robots is presented in Section 2. The proposed control scheme is described in Section 3. Some experiments and simulation results are presented in Section 4. Finally, our conclusions about this work are presented in Section 5.

## 2 Related Work

There has been a tremendous amount of research in the area of sensor-based control including sensor modeling, multisensor integration, and distributed control schemes for robotic applications in general and mobile robots in particular.

A sensor-based control using a general learning algorithm was suggested by Miller [2]. This approach uses a learning controller that learns to reproduce the relationship between the sensor outputs and the system command variables. Another technique for sensor-based obstruction avoidance for mobile robots was proposed by Ahluwalia and Hsu [3]. In their technique, the robot is able to move through an unknown environment while avoiding obstacles. Simulations were carried out assuming the robot had eight tactile sensors and the world is modeled as a two-dimensional occupancy matrix with 0's representing empty cells and 1's representing occupied cells. Another method for sensor-based obstruction avoidance was proposed by Gourley and Trivedi [4] using a quick and efficient algorithm for obstacle avoidance.

Hagar proposed a novel approach for sensor-based decision making system [5]. His approach

is based on formulating and solving large systems of parametric constraints. These constraints describe both the sensor data model and the criteria for correct decisions about the data.

There has been a fair amount of research in developing languages for sensor-based control for robot manipulators. The goal of such languages is to provide an easy tool for writing adaptive robotic controller. Some of these languages are described in [6]. Several research activities for sensor-based control for robotic applications can be found in [7].

Lin and Tummala [8] described an adaptive sensor integration mechanism for mobile robot navigation. They divided the navigation process into three phases:

**Sensing:** firing different sensors then sending the perceived data to the data processor.

**Integration:** interpreting sensory data of different types into a uniform representation.

**Decision:** Deciding the action plan based on the current workspace representation.

Luo and Kay [9] conducted a survey on multisensor-based mobile robots. In their survey, they presented a number of control strategies that has been used in this area.

A distributed decentralized control scheme is proposed by Mutambara and Durrant-Whyte [10]. This scheme provides flexible, modular and scalable robot control network. This scheme uses a non-fully connected control components, which reduces the number of interconnections and thus reducing the number of required communication channels.

The idea of *smart sensing* was investigated by several researchers. Yakovlev et al. [11] represented a dual purpose interpretation for sensory information; one for collision avoidance (reactive control), and the other for path planning (navigation). The selection between the two interpretation is dynamic depending on the positions and velocities of the objects in the

environment. Budenske and Gini [12] addressed the problem of navigating a robot through an unknown environment, and the need for multiple algorithms and multiple sensing strategies for different situations.

Discrete Event Systems (DES) is used as a platform for modeling the robot behaviors and tasks, and to represent the possible events and the actions to be taken for each event. A framework for modeling robotic behaviors and tasks using DES formalism was proposed by Košecká et al. [13]. In this framework, there are two kinds of scenarios. In the first one, reactive behaviors directly connects observations (sensor readings) with actions. In the second, observations are implicitly connected with actions through an observer.

In our proposed control scheme, the sensory system can be viewed as passive or *dumb* element which provides raw data. It can be viewed as an *intelligent* element which returns some “analyzed” information. Finally it can be vided as a *commanding* element which sends commands to the physical system. Each of these views is used in different situations and for different tasks. A detailed description of the proposed control scheme is presented in the following section.

### 3 The Proposed Control Scheme

The robot behavior can be described as a function  $\mathcal{F}$  that maps a set of events  $\mathcal{E}$  to a set of actions  $\mathcal{A}$ . This can be expressed as:

$$\mathcal{F}: \mathcal{E} \longrightarrow \mathcal{A}$$

The task of the robot controller is to realize this behavior. In general we can define the

controller as a set of pairs:

$$\{(e_1, a_1), (e_2, a_2), \dots, (e_n, a_n)\}$$

where  $e_i \in \mathcal{E}$ , and  $a_i \in \mathcal{A}$

The events can be defined as the interpretation of the raw data perceived by the sensors.

Let's define the function  $\mathcal{T}$  which maps raw data  $\mathcal{R}$  to events  $\mathcal{E}$ :

$$\mathcal{T}: \mathcal{R} \longrightarrow \mathcal{E}$$

The functions  $\mathcal{T}$  and  $\mathcal{F}$  can be closed form equations, lookup tables, or inference engine of an expert system. This depends on the kind of application and the complexity of each transformation.

### 3.1 Abstract Sensor Model

We can view the sensory system using three different levels of abstractions (see Figure 2.)

1. **Dumb sensor:** which returns raw data without any interpretation. For example, a range sensor might return a real number representing the distance to an object in inches, and a camera may return an integer matrix representing the intensity levels of each pixel in the image.
2. **Intelligent sensor:** which interprets the raw data into an event using the function  $\mathcal{T}$ . For example, the sensor might return something like “will hit an object,” or “a can of Coke is found.”
3. **Controlling sensor:** which can issue commands based on the received events. for example, the sensor may issue the command “stop” or “turn left” when it finds an

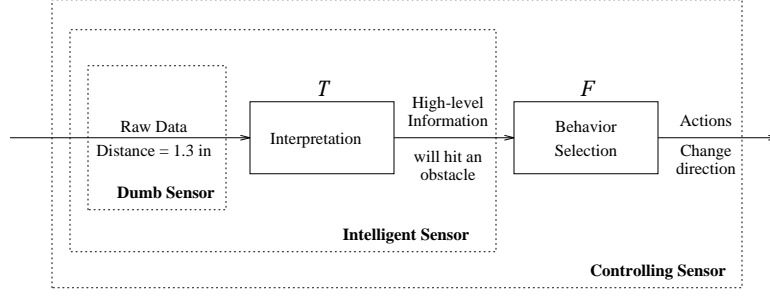


Figure 2: Three levels to view a sensor module.

obstacle ahead. In this case, the functions  $\mathcal{F}$  and  $\mathcal{T}$  should be included in the abstract model of the sensor.

The dumb sensor can be used as a source for the feedback information required by the control system. It can be also used to gather measurements to construct a map for the surrounding environment. The process that uses a dumb sensor as a source of information needs to know the type of that sensor, the format of the data the sensor returns, and the location of the sensor, to be able to interpret the perceived data. The intelligent sensor may be used for monitoring activities. The process that uses an intelligent sensor. needs to know only the event domain and maybe the location of the sensor. On the other hand, the commanding sensor is considered to be a “client” process that issues commands to the system.

### 3.2 A Distributed Control Architecture

Several sensors can be grouped together representing a logical sensor [14, 15]. We will assume that each logical sensor is represented as a client process which sends commands through a chanel to a multiplexer (the server process) which decides the command to be executed first. Besides these logical sensors, we might have other processes (general controllers) that send



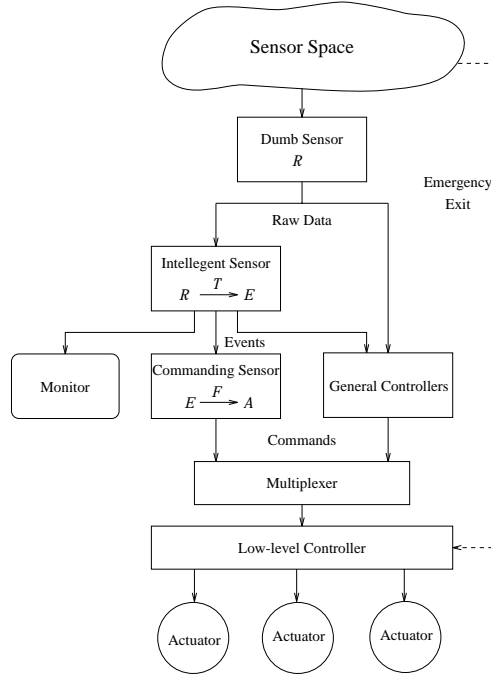


Figure 3: The proposed control scheme.

commands to the server process to carry out some global goals. Figure 3 shows a schematic diagram for the proposed control scheme.

Let's call any process that issues commands to the server a *client process*. In this figure, there are three types of clients:

1. Commanding sensors, that are usually used for reaction control and collision avoidance.
2. General Controllers, that carry out a general goal to be achieved (e.g., navigating from one position to another.)
3. Emergency exits, which bypass the multiplexer in case of emergencies (e.g., emergency stop when hitting an obstacle.)

In most cases, the general controllers require feedback information to update their control parameters. This information is supplied by dumb sensors in form of raw data, or by intelligent sensors in form of events. On the other hand, a monitoring process might use only intelligent sensors as a source of “high-level” events instead of raw data. All clients (except for the emergency exists) send the commands to a multiplexer. The multiplexer selects the command to be executed based on a priority scheme which depends on the current state of the system and the type of operation the client is performing. Once a command is selected, all other commands can be ignored, since the state of the system will change after executing the selected command.

The low-level controller, shown in Figure 3, translates the high-level commands into low-level instructions which drive the system’s actuators. The low-level controller receives its commands either form the multiplexer or from an emergency exit. After the command is executed, the system state is updated, and the sensor space is changed. New sensor readings are received and the cycle is repeated.

### **3.3 Communication Protocols**

In the proposed control scheme, there are several clients sending commands asynchronously to the server. Therefore, we need to define a communication protocol to organize these commands, and to set a priority scheme for selecting the command to be executed first. In most cases, the clients need to know the current state of the system and the command history to update their control strategy. Therefore, the server has to broadcast the selected command and the current state of the system.

Each client may send commands to the server (through multiplexer) at any time. Each

command is associated with the signature of the sender. This signature includes the name and type of the sender, and the priority value. In most cases, the reaction commands (usually from a commanding sensor to avoid collision) has a higher priority than any other client. The priority among the client may be specified by the user and/or by the current state of the system. Emergency exits should always bypass the multiplexer and sends its commands directly to the low-level controller.

The message passing paradigm is used for process communication. This allows processes to be running on different platforms without the need for shared memory. In our implementation, MPI, Message-Passing Interface [16] was used because of its portability and to workstation clusters and heterogenous networks of workstations. It also provides an easy-to-use library functions to carry out the required communication protocols.

### 3.4 Time vs. Accuracy

The most important criteria in any sensory system are *time* and *accuracy*. Time is the time elapsed between issuing a read request to the logical sensor and the reply to that request. This time depends on the physical aspects of the sensory system, and on the sensing strategy implemented in the logical sensor. Tolerance is defined in this scheme as the region in which the measurement resides.

The following are some variables that will be used in the tolerance analysis for our experiment.

- $v_s$  : sound velocity.
- $y_{max}$  : maximum distance in our indoor environment.

- $y_{min}$  : minimum distance in our indoor environment.
- $t_m$  : the maximum time to get a measurement by the physical sonar sensor.

$$t_m = 2y_{max}/v_s$$

- $v_r$  : the linear velocity of the robot in meter/sec.
- $\omega_r$  : the angular velocity of the robot in rad/sec.
- $t_d$  : decision time; the time to decide the next action based on the current reading.

In most cases, we cannot satisfy both requirement at the same time. Since the physical sensor has its accuracy limitations, therefore, we might need to get several readings regarding the same measured point to increase the accuracy. This of course with increase the time of measurement. In case of multisensor system, the accuracy can be increased by considering the readings from more than one sensor. In such cases, we should consider the time of the data fusion algorithms used.

## 4 Experiments and Simulation Results

A simulator called *XSim* has been developed to examine the applicability of the proposed control scheme. This simulator is based on a mobile robot called “LABMATE” designed by Transitions Research Corporation [17]. This simulator displays the robot on the screen and accepts actual LABMATE commands like *go*, *turn*, *read-sonars*, etc. In this environment, moving from the simulation to the real robot is simply a matter of compiling the driver program with the LABMATE library rather than the simulation library.



Figure 4: The LABMATE robot with its equipments.

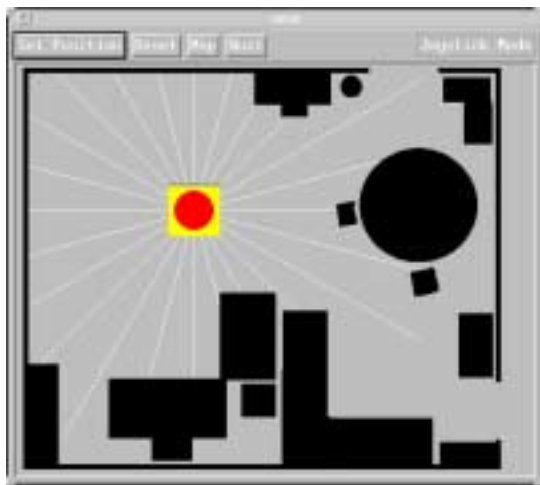


Figure 5: A graphical simulator for the LABMATE.

The LABMATE was used for several experiments at the Department of Computer Science, University of Utah. It also entered the 1994 AAAI Robot Competition [18]. For that purpose, the LABMATE was equipped with 24 sonar sensors, eight infrared sensors, a camera and a speaker.<sup>1</sup> Figure 4 shows the LABMATE with its equipment, and Figure 5 shows the graphical simulator for the LABMATE.

In all previous experiments, the LABMATE was controlled using a conventional control

---

<sup>1</sup>The LABMATE preparations, the sensory equipments, and the software and hardware controllers were done by L. Schenkat and L. Veigel at the Department of Computer Science, University of Utah.

strategy in which there is a central process (the controller) that does everything. This controller receives raw data from the “dumb” sensors, interprets the data, plans for the next move based on these readings and on the global goal it has to achieve, Tries to avoid obstacles, and finally issues the required commands. Beside that, the central controller may also produce an output for monitoring purposes. The following are some drawbacks for this scheme:

- The central controller has to know the type and location of each sensor.
- It also needs to know the data format for each sensor type.
- It may take long time to issue the required command. This time depends on the interpretation procedure for the data received from each sensor, and on the time to select the next command.
- Adding or removing any sensor requires modifying the central controller.

## 4.1 Modeling the System

The sensors in the old scheme are used only as dumb sensors, while in the proposed scheme, sensors are used in three different levels. They are used as dumb sensors to provide feedback information for a general navigator. They are also used as intelligent sensors providing information to a monitoring process (e.g., a speaker as an output device.) Finally they are used as commanding sensors (clients) for collision avoidance. The emergency exits are hardware bumpers that command the robot to stop if it touch any object. There is also a general controller for navigation and map construction. The commands that can be issued are:

- **GO-FRWD**  $d$ : move forward distance  $d$  inches, where  $d$  is a non-negative real number.

When  $d = 0$ , the robot will keep moving forward until other command is issued.

- **GO-BKWD**  $d$ : move backward distance  $d$  inches, where  $d$  is a non-negative real number.

When  $d = 0$ , the robot will keep moving backward until other command is issued.

- **TURN-RIGHT**  $\theta$ : turn right  $\theta$  degrees, where  $\theta$  is a positive real number.

- **TURN-LEFT**  $\theta$ : turn left  $\theta$  degrees, where  $\theta$  is a positive real number.

- **STOP**: stop moving (or turning).

- **RESET**: restart operation after a fault.

- **READ-SONAR**: read the sonar data.

- **GET-POSITION**: get the current position of the robot.

The system can be in any of the following states:

- **IDLE**: the robot is not moving.
- **FORWARD**: the robot is moving forward.
- **BACKWARD**: the robot is moving backward.
- **RIGHT**: the robot is turning right.
- **LEFT**: the robot is turning left.
- **FAULT**: the robot hit an obstacle.

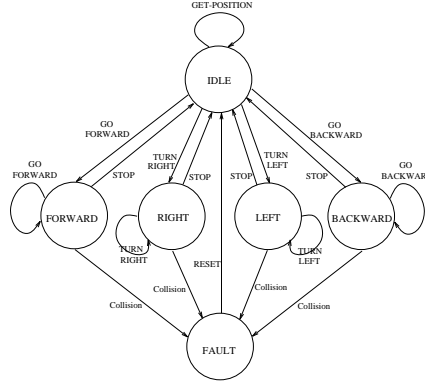


Figure 6: The relation between the system states and the commands.

Figure 6 shows a state diagram for the system. This figure shows that the robot has to go to the idle state when the command is changed. For example, if the command *GO-FORWARD* is issued, the system will go to the *FORWARD* state and will remain there as long as the following commands are *GO-FORWARD*. Once the next command is different, the system will go to the *IDLE* state first, then it will go to the state corresponding to the current command. This is analogous to what happens in controlling the LABMATE. The LABMATE has to stop first before changing direction. Notice that the command *READ-SONAR* is not present in that figure since it can be executed at any state.

## 4.2 Commanding Sensors and Reaction Control

To simplify our model, the 24 sonar sensors are divided into four logical sensors as shown in Figure 7.

1. **LS-FRWD** consists of the front 6 sensors.
2. **LS-BKWD** consists of the rear 6 sensors.



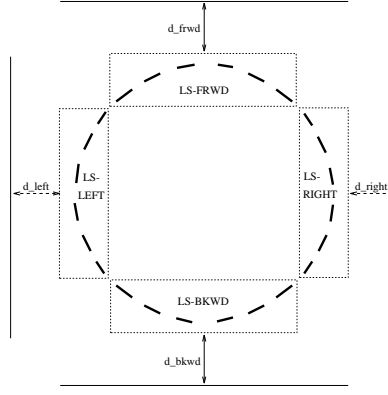


Figure 7: Dividing the sonar sensors into four logical sensors.

3. **LS-RIGHT** consists of the right 6 sensors.

4. **LS-LEFT** consists of the left 6 sensors.

These logical sensors communicate with each other to decide the command to be issued. This makes the job of the multiplexer easier, since it will deal with the four logical sensors as one client. The goal of the reactive control in this experiment is two fold:

1. Avoid obstacles.
2. Keep the robot in the middle of hallways, specially when moving through narrow corridors.

We will define two abstract values: *close* ( $c$ ) and *far* ( $f$ ). These two values represent the distance between the robot and the closest object at any of the four sides. The range for  $c$  and  $f$  are usually user defined values. The command to be issued as a reaction control depends on the current state of the system and the distance value at each side. There are several ways to define a command function  $\{$  to achieve the required goal. The assumption here is that there

is always enough space for the robot to rotate left or right, therefore there is no need to define any reaction control when the robot is rotating. One such function is shown in Table 1.

In this table, *TURN-L/R* means the command can be either *TURN-LEFT* or *TURN-RIGHT*, and a dash “—” means no command is issued. Notice that, in case of *d\_left* and *d\_right* have different values, the values for *d\_frwd* and *d\_bkwd* are not important. This is because we need to balance the distance to the left and to the right of the robot, and if, for example, the distance in front (*d\_frwd*) is *c*, and the robot state is *FORWARD*, then moving to the left (or to the right) will serve both; avoiding the object in front, and balancing the distance on both sides. In the first case of the table, when the distance is *c* in all sides, the robot will not be able to move anywhere, and the sensor readings will not change. This will result in a deadlock which requires external help by moving at least one of the obstacles for the robot to be able to move. Figure 8 shows graphically the different cases when the system state is *FORWARD*, and Figure 9 shows the same cases when the system state is *BACKWARD*.

### 4.3 The Priority Scheme

In this system, there are several clients for the server. Beside these clients, there are two emergency exits represented by two bumpers, one on the front and one on the back. As mentioned before, emergency exits do not compete for the server, rather it sends its commands directly to the low-level controller.

The priority scheme in our application is set by each client as a number from 1 to 10, with 1 as the highest priority. Normally, 1 is reserved for the collision avoidance client. The server checks for the priority associated with each command, and executes the command with the

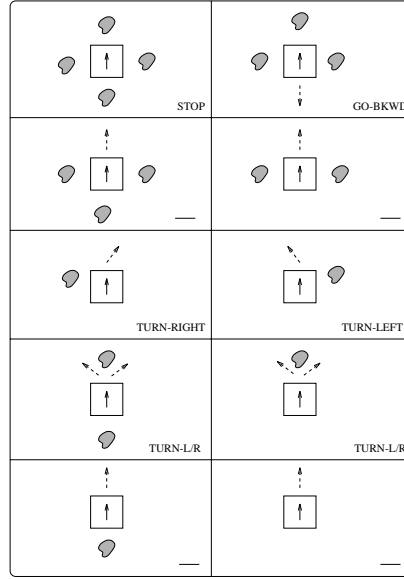


Figure 8: The reaction control when the system state = *FORWARD*.

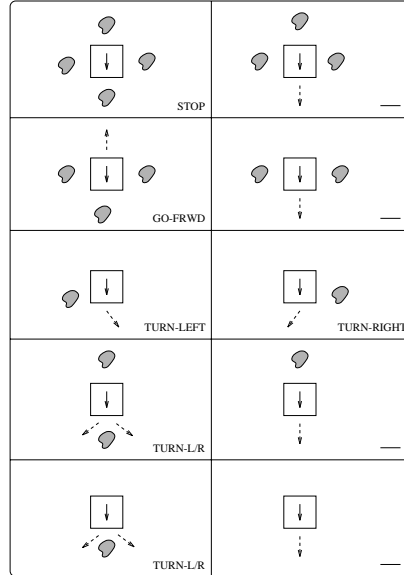


Figure 9: The reaction control when the system state = *BACKWARD*.

highest priority while notifying the “losers” which command was executed. If two commands with the same priority arrive at the same time, the server arbitrarily selects one of them and ignores the other.

Commands that were not selected are cleared since the state of the robot has been changed after executing the command with the highest priority.

#### **4.4 Simulation Results**

Several experiments were performed on the simulator to check the applicability and validity of the proposed control scheme, and the results were very encouraging. The following is a description of three of these experiments along with the output of the simulation showing the portion of the commands that were selected and the trajectory of the robot during each experiment.

##### **Experiment (1)**

This was the first experiment performed to demonstrate the applicability of this control scheme. In this experiment, two clients were running simultaneously; the collision avoidance client, and a simple navigator which always sends the command GO-FRWD. The collision avoidance has priority 1, which is the highest priority, and the navigation client has priority 9. The following shows part of the output printed during this experiment which shows the commands that have been executed by the server and some other information about the server activities.

Collision Avoidance: client #1.

Simple Navigation: client #2.

Server Starts as process #0.

\* Accepted RESET from 1 \*

- Rejected RESET from 1 \*

\* Accepted GO-FRWD from 2 \*

\* Accepted GO-FRWD from 2 \*

\* Accepted GO-FRWD from 2 \*

\* Accepted GO-FRWD from 2 \*

\* Accepted GO-FRWD from 2 \*

\* Accepted GO-FRWD from 2 \*

\* Accepted TURN-LEFT from 1 \*

- Rejected GO-FRWD from 2 -

\* Accepted TURN-LEFT from 1 \*

- Rejected GO-FRWD from 2 -

\* Accepted GO-FRWD from 2 \*

\* Accepted GO-FRWD from 2 \*

. . .

Two indoor configurations were used for these experiments; one representing a lab with

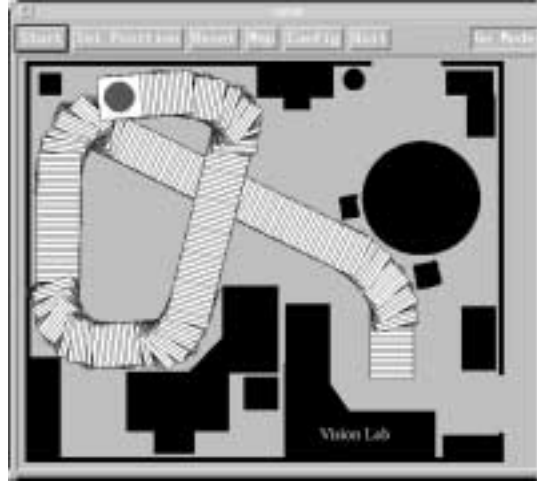


Figure 10: The trajectory of the robot in the lab environment.

tables and chairs, while the other represents long halls with doors and some obstacles. Figure 10 shows the trajectory of the robot in the lab environment, and Figure 11 shows the trajectory of the robot under the same experiment in the hallway environment.

## Experiment (2)

In the second experiment, we added another goal-directed client which tries to move the robot to a certain goal location. This client has priority 5 which is higher than the simple navigator process. This new client sends commands to the server to update the direction of the robot such that it moves towards the goal location. In this experiment, the initial and the final points were chosen such that there are some obstacles between them. Figure 12 shows the robot trajectory for this experiment from the initial location to the goal location. Notice that at several points, the collision avoidance client took over and moved the robot away from the obstacles, then the new client updates the direction towards the goal point.

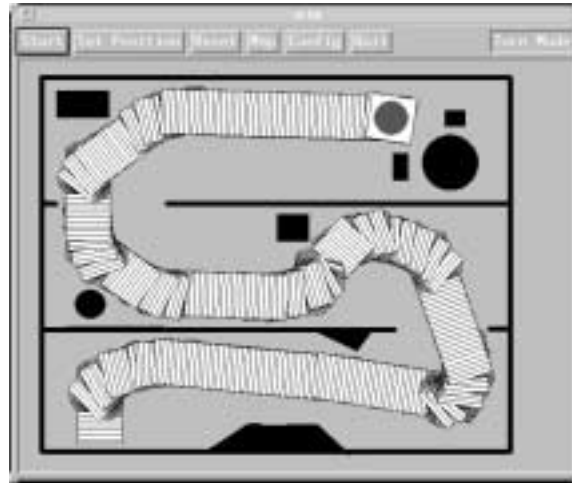


Figure 11: The trajectory of the robot in the hallway environment.

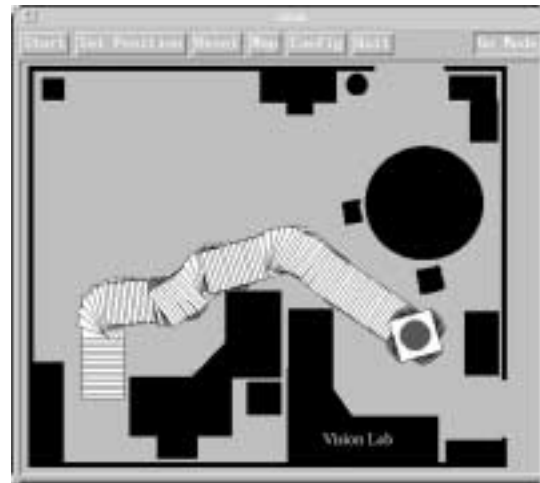


Figure 12: The trajectory of the robot from the initial to the goal point.

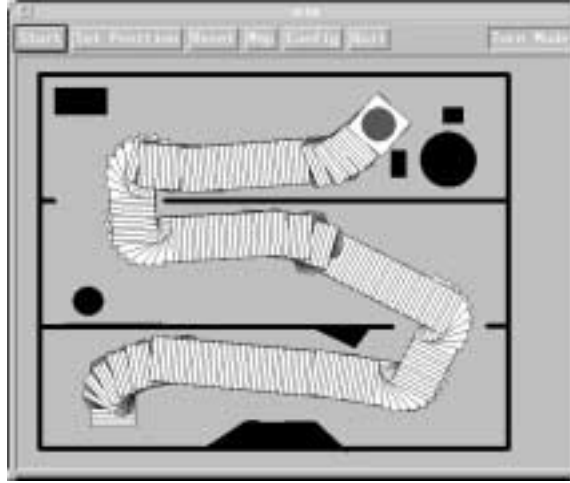


Figure 13: The trajectory of the robot while moving through open doors.

### Experiment (3)

In the third experiment, we replaced the goal-directed client with a door-finding client. This new client tries to find open doors and direct the robot to go through these doors. Finding doors using sonar sensor is very hard and problematic, and there is a lot of research in this area. For this experiment we used a very crude algorithm and a simple hallway structures just to demonstrate the capabilities of the proposed control scheme. Figure 13 shows the robot trajectory while moving in a hallway environment with two open doors at different places.

## 5 Conclusions

In this paper, a distributed sensor-based control scheme was proposed. In this scheme, each sensor can be viewed with three different levels of abstraction; *dumb sensors* which provide raw data, *intelligent sensors* which provides high level information in a form of events, and



finally, *commanding sensors* which can issue commands representing a reaction behavior for the system. Commands can be issued by different processes called *clients*. Each client may issue commands at any time, and a multiplexer (the server) selects the command to be executed. A priority scheme has to be defined as a bases for selection. An example for applying this control scheme to a mobile robot was described along with the positive simulation results. The logical sensor approach built on the *dumb sensory system* of our mobile robot allows a good flexibility and design modularity of controllers. By allowing for several levels of data abstraction and tolerance analysis based on the sensor type and required tasks, there is significant space for expansions. The need for multiple algorithms and sensing strategies based on situations and requirements it is reasonably lowered by using a distributed /logic control strategy. While other existing techniques are mainly evolving on specific sensing and control applications, the server/client parallel intelligent sensing approach proves a sufficient generic basis to allow divers and efficient controllers, and possibly nesting of various controllers. As immediate future steps would be a more detailed decision function for logical sensors, an explicit definition of the sonar sensors inter-communication protocols, and possibly higher level functions for increasing the accuracy of the measured point locations based on the different approaches discussed in the paper. The data noise could be also considered and modeled. We believe that this control scheme provides more flexible and robust control systems, and allows more modular design for the control systems. It also provides fast response for reaction behavior which is an essential requirement in real-time systems.

## References

- [1] M. Dekhil, G. Gopalakrishnan, and T. C. Henderson. Modeling and verification of distributed control scheme for mobile robots. Technical Report UUCS-95-004, University of Utah, April 1995.
- [2] W. T. Miller. Sensor-based control of robotic manipulators using a general learning algorithm. *IEEE Journal of Robotics and Automation*, pages pp. 157–165, November 1987.
- [3] R. S. Ahluwalia and E. Y. Hsu. Sensor-based obstruction avoidance technique for a mobile robot. *Journal of Robotic Systems*, 1(4):pp. 331–350, Winter 1984.
- [4] C. Gourley and M. Trivedi. Sensor-based obstacle avoidance and mapping for fast mobile robots. In *IEEE Int. Conf. Robotics and Automation*, 1994.
- [5] G. D. Hagar. Task-directed computation of qualitative decisions from sensor data. *IEEE Trans. Robotics and Automation*, 10(4):pp. 415–429, August 1994.
- [6] U. Rembold and K. Hormann. *Languages for Sensor-Based Control in Robotics*. Springer-Verlag, 1987.
- [7] C. S. George Lee. *Sensor-based robots: algorithms and architecture*. Springer-Verlag, 1991.
- [8] C. C. Lin and R. L. Tummala. Adaptive sensor integration for mobile robot navigation. In *IEEE International Conference on Multisensor Fusion and Integration*, October 1994.
- [9] R. C. Luo and M. G. Kay. *Multisensor integration and fusion for intelligent machines and systems*. Ablex Publishing Corporation, 1995.

- [10] A. G. O. Mutambara and H. F. Durrant-Whyte. Modular scalable robot control. In *IEEE International Conference on Multisensor Fusion and Integration*, October 1994.
- [11] A. Yakovleff, X. T. Nguyen, A. Bouzerdoun, A. Moini, R. E. Bogner, and K. Eshraghian. Dual-purpose interpretation of sensory information. In *IEEE Int. Conf. Robotics and Automation*, 1994.
- [12] J. Budenske and M. Gini. Why is it difficult for a robot to pass through a doorway using ultrasonic sensors? In *IEEE Int. Conf. Robotics and Automation*, pages 3124–3129, May 1994.
- [13] J. Košecká and L. Bogoni. Application of discrete event systems for modeling and controlling robotic agents. In *IEEE Int. Conf. Robotics and Automation*, pages 2557–2562, May 1994.
- [14] T. C. Henderson and E. Shilcrat. Logical sensor systems. *Journal of Robotic Systems*, pages pp. 169–193, March 1984.
- [15] E. D. Shilcrat. Logical sensor systems. Master’s thesis, University of Utah, August 1984.
- [16] University of Tennessee, Knoxville. *MPI: a message-passing interface standard.*, May 1994.
- [17] TRC Transition Research Corporation. *LABMATE user manual, version 5.21L-f.*, 1991.
- [18] L. Schenkat, L. Veigel, and T. C. Henderson. Egor: Design, development, implementation – an entry in the 1994 AAAI robot competition. Technical Report UUCS-94-034, University of Utah, December 1994.

<i>d_right</i>	<i>d_left</i>	<i>d_frwd</i>	<i>d_bkwd</i>	<i>FORWARD</i>	<i>BACKWARD</i>
<i>c</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>STOP</i>	<i>STOP</i>
<i>c</i>	<i>c</i>	<i>c</i>	<i>f</i>	<i>GO-BKWD</i>	—
<i>c</i>	<i>c</i>	<i>f</i>	<i>c</i>	—	<i>GO-FRWD</i>
<i>c</i>	<i>c</i>	<i>f</i>	<i>f</i>	—	—
<i>c</i>	<i>f</i>	<i>c</i>	<i>c</i>	<i>TURN-RIGHT</i>	<i>TURN-LEFT</i>
<i>c</i>	<i>f</i>	<i>c</i>	<i>f</i>	<i>TURN-RIGHT</i>	<i>TURN-LEFT</i>
<i>c</i>	<i>f</i>	<i>f</i>	<i>c</i>	<i>TURN-RIGHT</i>	<i>TURN-LEFT</i>
<i>c</i>	<i>f</i>	<i>f</i>	<i>f</i>	<i>TURN-RIGHT</i>	<i>TURN-LEFT</i>
<i>f</i>	<i>c</i>	<i>c</i>	<i>c</i>	<i>TURN-LEFT</i>	<i>TURN-RIGHT</i>
<i>f</i>	<i>c</i>	<i>c</i>	<i>f</i>	<i>TURN-LEFT</i>	<i>TURN-RIGHT</i>
<i>f</i>	<i>c</i>	<i>f</i>	<i>c</i>	<i>TURN-LEFT</i>	<i>TURN-RIGHT</i>
<i>f</i>	<i>c</i>	<i>f</i>	<i>f</i>	<i>TURN-LEFT</i>	<i>TURN-RIGHT</i>
<i>f</i>	<i>f</i>	<i>c</i>	<i>c</i>	<i>TURN-L/R</i>	<i>TURN-L/R</i>
<i>f</i>	<i>f</i>	<i>c</i>	<i>f</i>	<i>TURN-L/R</i>	—
<i>f</i>	<i>f</i>	<i>f</i>	<i>c</i>	—	<i>TURN-L/R</i>
<i>f</i>	<i>f</i>	<i>f</i>	<i>f</i>	—	—

Table 1: An example of a decision function for reaction control.